实验 8　C#解析 XML 文件实验

【实验内容提要】

1.解析 XML 文件有哪些方法？各有什么优缺点？

2.XPath 解析 xml 文档的要点。

在程序中访问并操作 XML 文件一般有两种模型：流模型和 DOM（文档对象模型）。流模型中有两种变体——“推”模型和“拉”模型。

“推”模型也就是常说的 SAX，SAX 是一种靠事件驱动的模型。它每发现一个节点就用“推”模型引发一个事件，而我们必须编写这些事件的处理程序，很麻烦。

.NET 中使用的是基于“拉”模型的实现方案。“拉”模型在遍历文档时会把感兴趣的文档部分从读取器中拉出，不需要引发事件，允许我们以编程的方式访问文档，这大大的提高了灵活性，“拉”模型可以选择性的处理节点。在.NET 中，“拉”模型通过 XML 阅读器(XMLTextReader 类)来实现的。该类提供 Xml 文件读取的功能，它可以验证文档是否格式良好，如果不是格式良好的 Xml 文档，该类在读取过程中将会抛出 XmlException 异常。任何时候在内存中只有当前节点，但它是只读的，向前的，不能在文档中执行向后导航操作。

DOM 的好处在于它允许编辑和更新 XML 文档，可以随机访问文档中的数据，可以使用 XPath 查询。但是，DOM 的缺点在于它需要一次性的加载整个文档到内存中，对于大型的文档，这会造成资源问题。在.NET 中使用 XML DOM 分析器（XMLDocument）实现 DOM 模型。

因此，.NET Framework 完全支持 XML DOM 模式，但它不支持 SAX 模式。.NET Framework 支持两种不同的分析模式：XML DOM 分析器(XMLDocument 类)和 XML 阅读器(XMLTextReader 类)，不支持 SAX 分析器，但这并不意味着它没有提供类似 SAX 分析器的功能。通过 XML 阅读器可以将 SAX 的所有的功能很容易的实现及更有效的运用。

在项目中，我们选用 xpath 的方式来解析 xml 文档。这是基于以下的几点原因：

1、文件大小。要处理的文件不大，一般都在几百 K 到 1M。

2、 XPath 的灵活性。不需要获取文档的全部数据，只需要获取大部分想要的数据。

3、 学习代价低。符合一般的思维习惯，通过 Path 获取结果。

通过 XPath 的方式解析 xml 文档，需要先加载文档，然后再读取想要的节点值。

xml 文档类：protected　XmlDocument　doc =null;

xml 文档的根元素（节点）：protected　XmlElement　root =null;

xml 文档的名空间管理器 ：protected　XmlNamespaceManager　nsmgr =null;

接下来就是加载文档的代码：

```
protected void LoadXmlFile(FileInfo xmlFile)
    {
      if (xmlFile == null || !xmlFile.Exists)
      {
        throw new FileNotFoundException(string.Format("要解析的文件不存在{0}。
",xmlFile.FullName));
      }
      //加载文件
      this.doc = new XmlDocument();
      doc.Load(xmlFile.FullName);
      //准备读取文件
      root = doc.DocumentElement;
      string nameSpace = root.NamespaceURI;
      nsmgr = new XmlNamespaceManager(doc.NameTable);
      nsmgr.AddNamespace("ns", nameSpace);
    }
```

这里有几行要注意，这两行是取得 xml 文档的名空间：

```
root = doc.DocumentElement;
string nameSpace = root.NamespaceURI;
```

这两行是建立 xml 文档的名空间管理器：

```
nsmgr = new XmlNamespaceManager(doc.NameTable);
nsmgr.AddNamespace("ns", nameSpace);
```

如果你的 xml 文档有命名空间，则这部分的代码是必不可少的。

接下来就是读取文档节点的值。

这里两个传入的参数 prefixPath 是节点的上级节点路径，xRelativePath 是要读取的节点名称。另外，变量 XmlFileInfo 是要加载的 xml 文件。

```
protected string GetNodeValue(string prefixPath, string xRelativePath)
    {
        if (doc == null)
        {
            LoadXmlFile(XmlFileInfo);
        }
        string xPath = string.Empty;
        if (!string.IsNullOrEmpty(xRelativePath))
        {
            if (!string.IsNullOrEmpty(prefixPath))
            {
                xPath = prefixPath + xRelativePath;
            }
            else
            {
                xPath = xRelativePath;
            }
        }
        xPath = xPath.Replace("/", "/ns:");
        XmlNode node = root.SelectSingleNode(xPath, nsmgr);
        if (node == null)
        {
            return null;
        }
        return node.InnerXml;
    }
```

为什么要设置两个参数 prefixPath 和 xRelativePath 呢，其实这个没有多大的关系，同学们也可以在方法外确定了这个 XPath，在方法中只设置一个传入参数，效果是一样的。

注意这一行：

```
xPath = xPath.Replace("/", "/ns:");
```

如果你的 xml 文档带命名空间，则这行是比不可少的，否则会出现找不到节点，无法解析的情况。

这里还有一个关于 XPath 的问题： 对于下面这样一个 xml 文档，要查找第一个节点下的学生的 Name 时（ID=01），其 XPath 应该是"/ns:Root/ns:Students/ns:Student[1]/ns:Name"。xml 对于重复的节点名称，是按照顺序 1，2，3...的方式遍历的，也就是说如果要找第 N 个 Student 节点的下的节点之，那么应使用 Student[N]的标识方式。

```
<?xml version="1.0" encoding="UTF-8" ?>
<Root xmlns="urn:ClassNameSpace">
<Class>
<ClassID>1234</ClassID>
</Class>
<Students>
<Student>
<ID>01</ID><Name>Name01</Name>
</Student>
<Student>
<ID>02</ID><Name>Name02</Name>
</Student>
</Students>
</Root>
```

当然，这里也可以获取节点属性的值，查找满足特定值的节点等等。

或者获取 students 下每个 student 的值：

```
private void button1_Click(object sender, EventArgs e)
        {
            XmlDocument XMLDom = new XmlDocument();
            XMLDom.Load("c:/class.xml");
            XmlNodeList newXMLNodes = XMLDom.SelectNodes("/root/s
tudents");
            foreach (XmlNode xn in newXMLNodes)
            {
                string title = xn.SelectSingleNode("student").Inn
erXml;
                MessageBox.Show(title);

            }
```

【实验】

请同学们参照上面的介绍，自行编写解析下面 xml 文档的 c#程序，要求将解析得到的结果打印在 dos 控制台上。

```
<?xml version="1.0" encoding="UTF-8" ?>
<Root xmlns="urn:ClassNameSpace">
<Class>
<ClassID>1234</ClassID>
</Class>
<Students>
<Student>
<ID>01</ID><Name>Name01</Name>
</Student>
<Student>
<ID>02</ID><Name>Name02</Name>
</Student>
</Students>
</Root>
```